Communication Time in Extreme ProgrammingTeams: A Controlled Case Study

Article · October 2015

CITATIONS
0

READS
9

5 authors, including:

Ali Aljadaa
Birzeit University
4 PUBLICATIONS **162** CITATIONS
SEE PROFILE

Abdallah Karakra
Birzeit University
2 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Abed Tahseen Othman
Birzeit University
2 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Adel Taweel
Birzeit University
89 PUBLICATIONS **429** CITATIONS
SEE PROFILE

Communication Time in Extreme Programming Teams: A Controlled Case Study

Ali A. Al-Jadaa, Ministry of Education, Palestine, ali.ps@live.com
Abdallah H. Karakra, Birzeit University, Palestine, akarakra@gmail.com
Mohammad ZeinEddin, Birzeit University, Palestine, mohammad@zeineddin.name
Abed Othman, Birzeit University, Palestine, abedothman83@gmail.com
Adel Taweel, Birzeit University, Palestine, adelt00@gmail.com

***Abstract*—Software development is a team-centered activity. However, there are a number of factors that affect a team's productivity. Thus, understanding the team dynamics and the different factors that influence their working is essential to improving their productivity. One such key factor is the amount of time a team spends on communicating to completing their tasks. This paper reports on a small scale case study, of (eight participants in) four student groups, using extreme programming. The study was conducted in a controlled environment to study the factors that may affect a team communication time, during the implementation phase. We found communication time between team members of the same gender is the longest. However, forming a team with relatively equal experience and including a well documented pre-training reduces the communication time significantly**

Keywords: extreme programming, communication time, communication overheads, teamwork.

I. Introduction

Teams and teamwork in organizations are essential for many types of domains and tasks, such as software development. It is reported as essential in management best practice lists and proposed as a basic attribute in employee recruitment decisions [6]. Agile programming methods suggest that programming must be conducted in teams, and one of the popular agile software development methodologies is extreme programming (XP) [8]. Employing extreme programming, for many types of tasks, is seen as one of the most effective process methods. Of particular relevance, that it attains customer satisfaction, keeps customers engaged, continually getting their feedback and comments for a better product. Development teams are able to show systems to the customers in the early stages for both to receive early feedback as well as better understand their requirements and respond to them promptly [8].

However, one key factor that influences team effectiveness is communication overhead between team members. It affects decisions related to how to form available XP team as well as how to select its members to achieve maximum productivity. There are many factors that affect communication between team members, given the proximity of working between each pair, within the team and with other parties. If these factors are well understood, they will contribute towards building a more effective and potential productive team. Factors that may influence communication time in XP teams, which are of interest in this paper, include members' gender, skills, experience, and age.

This paper is organized as the following: section 2 reports briefly on related work to our case study. Section 3 describes the case study design and the conducted experiment. Section 4 presents the data analysis of the collected data. Section 5 summarizes conclusions derived from the results. Section 6 notes the threats to the validity of the case study. And finally section 7 presents the recommendations.

II. Related Work

Extreme programming (XP) is a general and common software development method [1]. It focuses on the use of a teamwork to create a team organization for enabling teams to become highly productive [8]. Working in a team may be affected by several factors, these factors may improve and reduce the time needed to implement the project or may disrupt and potentially fail the project due to ineffective communication between the members of a team.

Several factors that influence communication time in extreme programming have been reported in the literature, including team membership, composition, structure, processes, psychology, tasks and task design, as well as organizational context, resources, structure and environment [3]. Harrison et al. [5] and Chatman et al. [2], demonstrate that demographic differences are initially more powerful than cognitive differences (when groups first form). However, research on demographic similarities in teams is not as regular [4]. Williams et al. [9] indicate that demographic dissimilarities do matter in team formation. After the meta-analysis of the last 40 years of studies and research, they found that the variety in characteristic and features such as age, gender and tenure have a negative effect on team communication. Wagner et al. [7], argue that the team with similar age and similar social and economic experiences have positive effects on team communication. However, some studies reported ambiguous and conflicting results, on software development, in relation to the similarity and diversity in gender. Gibson and Zellmer-Bruhn in [3], demonstrate that similarities in age, tenure and gender, between the members of the team decreases conflict and facilitates the social integration, as well as it contributes to performance and stability of team activities.

LOC	
Level	*Weight*
LOC<5000	1
5000-10000	3
LOC>10000	5

Education	
Specialization	*Weight*
Computer Science	5
Computer Engineering	5
Engineering	3

Job	
Title	*Weight*
Developer	5
Trainer	4
Teacher	3
Support	3

Experience Years	
Years	*Weight*
Years >3	5
Years = 2	3
Years <1	1

TABLE I
WEIGHT OF LEVELS

III. CASE STUDY AND DATA COLLECTION

The case study reported on this paper was part of the Software Engineering course in the Master in Computing program at Birzeit University, and it was held in the first semester of 2014/2015. The software task of the experiment was to design and develop an extended calculator, able to perform, in addition to the basic arithmetic operations, several more sophisticated functions, including a currency converter, and a salary and tax calculator.

The study was conducted in three phases. In the first phase, preparation phase, potential participants were asked to fill a questionnaire, which contained questions about their age, gender, qualification and specialization, their preferred team size and gender, social activities they may undertake during programming, and their programming experience and the programming languages they know and/or have experience with. Based on these, 8 participants were selected and grouped into 4 XP teams, where each team contains two participants. Participants grouping, or team formation, was based on their programming experience, the programming languages they know or have most experience with, their related social activities, and gender. To evaluate and determine a normalized level of experience for all participants, the following metric was used to determine the level of experience as :

- Number of lines of code (LOC) in programming languages (C,C#,JAVA,PHP, ...).
- The company or the organization of work, role of Job and the experience years.
- Education university and specialization.

Where the number of lines of code (LOC) divided into three levels, less than 5,000, from 5,000-10,000 and more than 10,000. And the experience years in every company or organization divided into four levels, less than two years, two years, three years and more than three years. Finally, the education university and specialization divided into two levels, the Computer Science, Computer Engineering, Engineering. Table 1 presents the weight for each level for each metric. To calculate the final experience for each participant we used these equations :

Years = $\sum Years * Weight/BestCase * 50\%$(1)

LOC = $\sum Weights/BestCase * 35\%$(2)

Education = Weight/Best Case*15% (3)

Total Experience = Years+ LOC+ Education (4)

The best case given 1, and we considered above 0.60 is high experienced, 0.35 to 0.60 is Mid and the less than 0.35 is low.

The resultant groups were as follows:

- T1-High experienced male with low experienced female.
- T2-Two middle experienced females.
- T3-Two middle experienced males .
- T4-High experienced male and female.

In the second phase, a system design, using service-oriented software engineering approach, was prepared, to use as a basis for a software task to study factors of interest during the implementation stage of software development. Participants of each team were assigned one of either two roles: one who writes the code, and the other reviews each line of code as it is typed in. In this phase, a dry run of the experiment was undertaken to ensure its smooth run.

In the third phase, execution phase, experiments was conducted, one for the each of the four participating groups. Custom forms were designed to collect data during the experiment and each group were assigned an observer to observe and record their actions while programming. The observer has used an excel spreadsheet to register the time, which was captured using a stop watch, to enable collecting information without affecting programmers behavior.

IV. DATA ANALYSIS

After conducting the experiments, from the time sheets of each team experiment, the details of the analysis were grouped based on the communication time that was spent by each team. The study distinguished between three overall types of time: implementation time, communication time and testing time.

A. Implementation Time

The actual programming time during the implementation is presented in Figure 1. This is the time that was actually spent by participants programming the system. To enable the observer precisely capture this time, it was captured in terms of the design of the software task and its implementation components or activities. The design included three main components (or classes) and/or programming activities of the calculator system: MathOperation classs, SimpleCalculatorGUI and, Integration between the two. Thus the implementation time consists of the time of programming the MathOperation class, the SimpleCalculatorGUI, and integrating the UI elements to the appropriate methods and services in between the two. The estimated programming time, based on the designer estimates, of this activity was 100 minutes, but actual programming time varied across different teams as shown in Figure 1.

As a general finding, when team members have almost similar levels of experience and use the same programming language, the time to program the project is shorter than those who have different levels. Figure 1 shows that highly experienced team members with lower experienced team members is not recommended since the highly experienced members

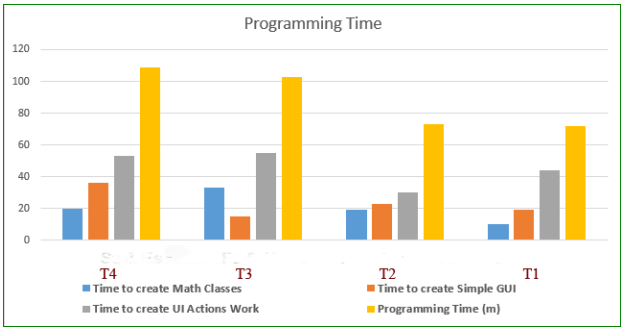

Fig. 1. Programming Time

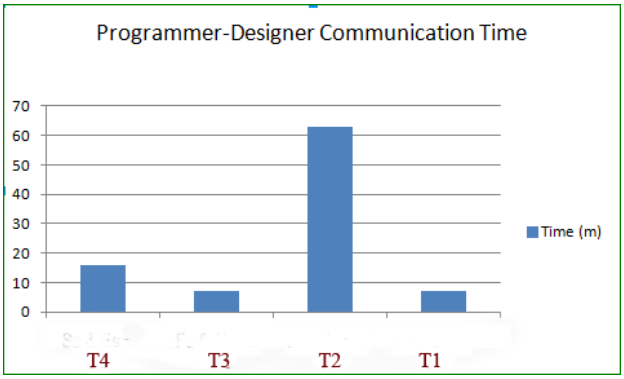

Fig. 2. Programmer-Designer communication time

will end up doing most of the work, as is the case in team T1. Also, it is noted that, in this experiment, providing a short pre-training to low-experienced participants, e.g. team T2, resulted in a better understanding of the system and reduced the gap of programming knowledge between participants, but this result could not be generalized that it was only provided to one group.

Figure 1 shows that the teams that have participants who, noted in their questionnaire's answers that, didn't like to work in teams, e.g. team T4, took longer time to complete the experiment compared to the other teams who liked to work within small teams.

B. Communication Time

This subsection presents the communication time that was recorded for activities related to communicating between team members, including times related to communication between programmer-designer and the times taken for eating, drinking, talking and searching the Internet. These times were seen as an essential communication time- i.e. contributing factors to communication between team members within a team, classified as personal and work time. The communication time was divided into four different types: programmer-designer, team-members, social and internet-search. These times were captured, in an excel spreadsheet, separately as columns, recording different type of each communication time during the experiment.

1) Programmer-Designer Communication Time: The programmer-designer communication time occurs when the programmer communicates with the system designer for questions regarding the system design itself, the project in general, or asks for help, for example, the programming language code.

As shown in figure 2, the teams who have low experience tend to communicate more with the designers compared to teams with higher experience. This can be explained that teams with higher experience have a higher ability to understand the design compared to those with lower experience. Team T2, which is a low-experience team has taken additional pre-training, which has been taken into consideration and thus was added to the programmer-designer time.

2) Team-Members Communication Time: This refers to the communication time spent between the team members to accomplish the software task. It only includes the time that members of each team needed to communicate to work on activities related to accomplishing the assigned software task.

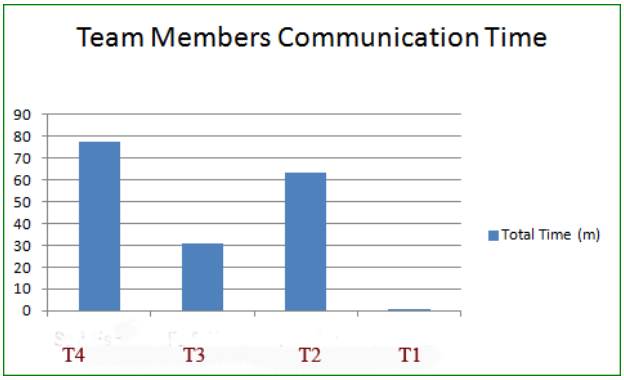

Fig. 3. Team Members Communication Time

We found in Figure 3 that the teams who have similar experience, communicate with each other in a more effective and efficient way, as is the case with teams T2 and T3. But for the teams with high experienced members, we found there is a longer communication time, which is often noted, because of differences in opinion of the members, as is the case in team T4. On the other hand, for the team with large differences in the level of experience of its members, there was almost no communication between the team members, e.g. T1, which could be explained by the low level of confidence of the low-experienced member and potentially by the domination of programming work by the experienced member.

3) Social: Eating, Drinking, and Talking Communication Time: This is the time spent by team members on activities not related to the software task itself, like having a break to eat, drink, making a personal phone call, talking to each others, talk on mobiles, or similar other things.

Figure 4 shows that the time spent on activities such as eating, making phone calls, browsing Facebook, etc. Have longer social-time, but with no major negative effects on the total communication time. On the contrary, we noticed from

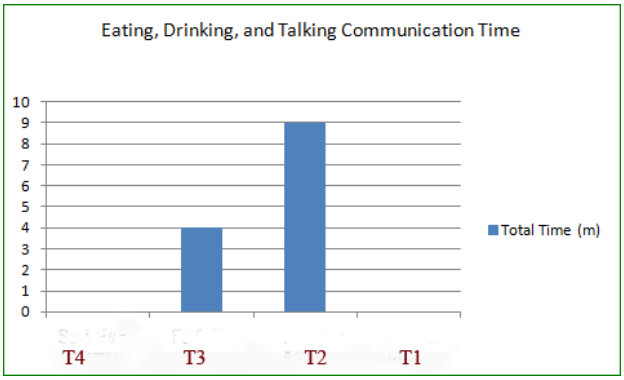

Fig. 4. Eating, Drinking, and Talking Communication Time

the observation, members who have similar common levels of understanding, they talk and browse Facebook without any negative effect on their total communication time, such as teams T2, and T3. On the other hand, we found the teams containing members with large differences in experience or are high-experienced members, have almost nil or very low social-time, effectively did not talk or did anything outside the experiment such as the teams T4 and T1.

Another thing to note is that the teams with the same gender tends to talk and do more of the social activities more than teams with different genders. This could be explained due to culture factors in which talking and working with a team member who has the same gender is much more comfortable than working with one with a different gender.

4) Internet-search Time: Figure 5 presents the internet-search time which is the time that is spent by the participants on the internet searching for project implementation, e.g. searching for language code.

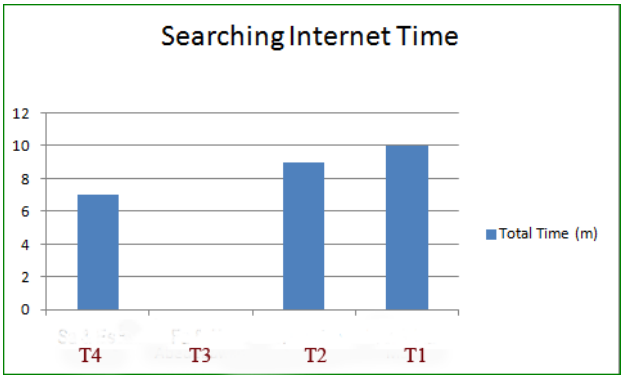

Fig. 5. Searching Internet Time

It is inevitable that programmers would need some help with the programming language during their programming, e.g. searching the internet for code, but unless one of the XP pair can provide an answer, then perhaps searching the internet may not be needed, as was the case for T3, which has not used the Internet for searching. The reason behind could be, as was observed, that T3 preferred asking the designer, who is more expert in the programing language, rather than searching the internet, which obviously affected designer-member time. For the T2, they used the search engine, but not significantly. This could also be explained by the pre- training that was provided to this group before the experiment, which may have contributed to their immediate needed programming knowledge.

C. Testing Time

To ensure uniformity across different teams, we needed to relate their overall programming time to how much they completed of the software task. Thus, this section presents the way that was used to evaluate the experiment time and its completion versus the number of use (or test) cases or requirements of the software task. There are two important aspects to ensure normalization of overall time of the experiment across all participants. The First aspect is to write the test cases before coding and then ask each team to run automatic testing scenarios on their programmed code and compare them against the test cases. Writing the test cases before coding is a substitute for specification, effectively noting functions, parameters and arguments for methods, and their testable results. This will give us an indication of how much each team completed the software task. To reach uniformity, or normalized results, across teams for a total experiment time, teams with failed use cases are asked to undertake the ”test-repair code” cycle until all test cases are successful. This will ensure all teams have completed the software task and produced (the same amount of) a working code, thus total experiment time is uniformly distributed across the four runs, for a normalized statistical comparison. The second aspect is to capture the time each team spends performing this testing task, and to observe the time each spends communicating during the ”test-repair code” cycle. All teams are asked to only undertake the test cases, or testing, towards the end after have finished the implementation. Due to its highly intertwined nature with programming, unit testing was not captured.

To capture this testing time each observer recorded the time it takes each team to test their developed software task, against the requirements (or test cases) as per the set specification. A testing plan was pre-created noting the task and weighting point for each task. Each was graded to ensure the time is normalized against the size of effort achieved during the recorded programming time.

From the results, we found the high experienced or trained teams tended to do testing in a shorter time. This could be explained that higher experienced teams code their program more carefully and robustly and so they were testing the application after every step thoroughly and potentially took them lesser time to fix errors at this end testing stage. On the other hand, for the teams with middle or no experience, the testing took longer to find bugs, errors...etc. And may have needed repeating the ”test-repair code” cycle more than once to fix these errors.

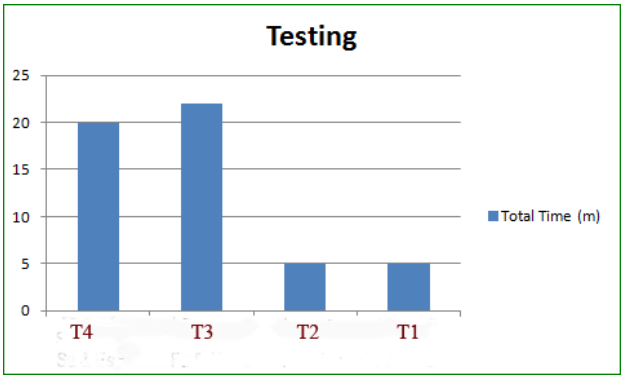

Fig. 6. Testing Time

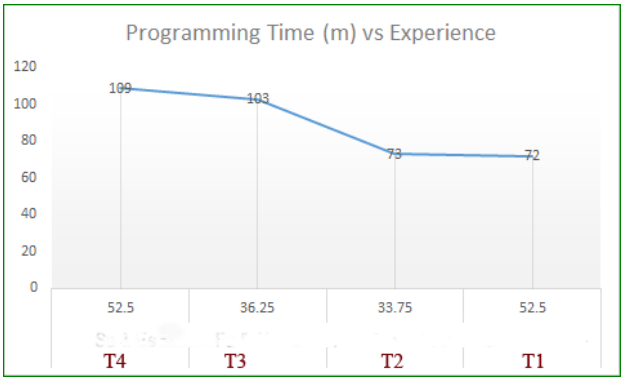

Fig. 7. Programming Time vs Experience

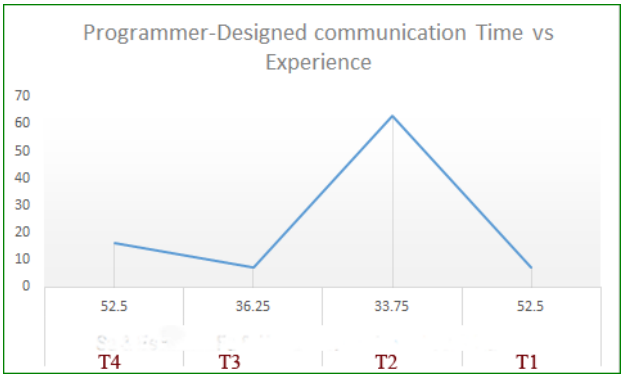

Fig. 8. Programmer-Designer communication time

V. Conclusions

The paper reports on a case study that was designed to study a set of factors that potentially may influence communication time in software development with a focus on extreme programming as a process model and the use of service-oriented software engineering as a design method. The study was run in a controlled setting using 8 participants divided across 4 XP teams to study communication time during the implementation phase. Although the study was a small study and a larger study would be needed to scale its results. However, we believe the study provides good indicative results that generally align with the existing literature.

As a first and major finding, we found that the experience of the implementer is the main factor that affects most parts of our experiment, as shown in the figures below. Other factors that play parts in communication time is the gender and the preferred team size, which affected some activities of the experiment.

Level of experience versus programming time: as noted above, such relation is anticipated that the more experience the team the shorter the programming time. Bus also, when one of the team members is a higher experience member then the level of experience of the second team member has no significant noted effect on programming time. Also, we noticed that a pre-training session to share the design and technology, between designer-programmer, before programming commences can raise the experience level of the programming team and that may particularly help members with lower experience. These results are shown in figure 7, where Ah has a high experience, Am and Ar has the training session, and the other 2 teams are of lesser experience in the experiment.

For the designer-members communication, Figure 7 shows that this time is higher for teams with lower experience, due to the need for greater explanation to understand the system and start the implementation.

For the team-members communication time, as shown in figure 8 we found that the teams with the same gender tend to communicate more than teams with different genders in general. As noted above, in this experiment this may be due to a result of the participants conservative culture in which communication between members of the same gender is more comfortable.

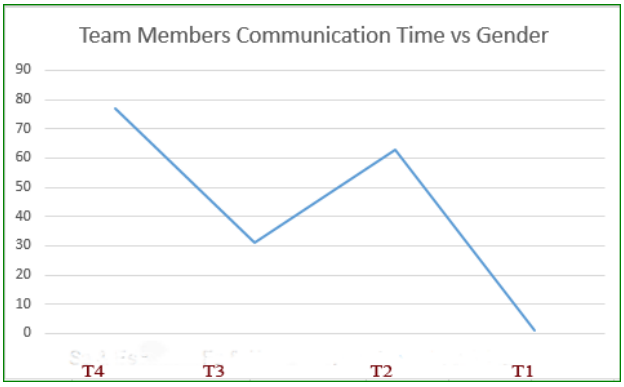

Fig. 9. Team Members Communication Time vs Gender

Similarly, social communication time with activities that are not related to the programming like eating, drinking, and mobile talking, the time of these activities is higher for teams with members of the same gender, and lowers for teams with members of different genders.

For the testing time, it is very much related to the level of experience, similar to the programming time, the testing time is lower for training or high experienced programmers, while it is higher for low experienced programmers. See figure 6.

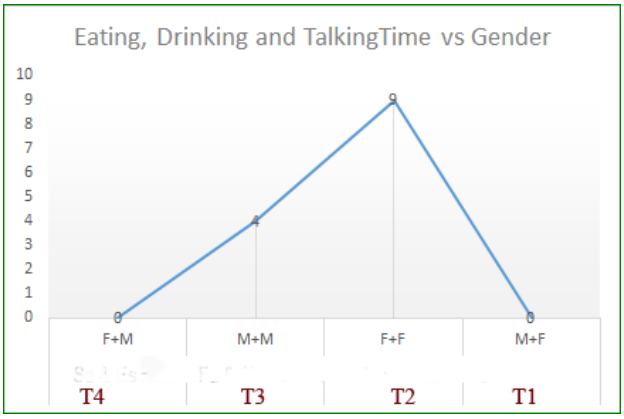

Fig. 10. Eating, Drinking, and Talking Communication Time

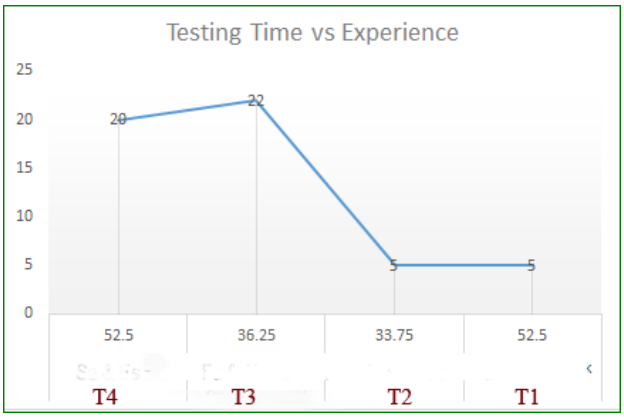

Fig. 11. Testing Time

Based on the above results, it thus may be more effective to select a team with members who have a roughly equal level of experience. In addition, members of the team may benefit from a pre-training, between design-programmer, before the implementation stage commences, as reported above this training may help improve their understanding of the design for the implementation of the system. Further, having XP teams with mixed genders may be more productive for conservative cultures, as this may reduce the overall communication time as team members may only communicate at work-related activities, although one cannot generalize from such a small observation, and would need further investigation. Selecting a team with high experience may reduce the testing time, since highest experience members may have developed techniques to produce a better quality code.

VI. Threats To Validity

For the case study only a set of factors were chosen to study, but we realize that there are other factors that could affect the communication time of the software implementation. Thus, in this study we studied the factors of interest that we could measure accurately and that we believe directly relate to communication time.

The study is a small scale study and we appreciate that a larger study would be needed to confirm some of these results, thus understandably could not be generalized. However, we believe, the reported results can be explained and do align with the literature.

In addition, it is worth noting that the time we collected while observing the participants is manually collected to as accurately as possible, but on occasions for some activities, which are intertwined, overlapping and/or run concurrently, measurements may have minor variations in accuracy. For example, when team members talk on a social subject while programming, these two concurrent activities could not be measured very precisely at the same time. In such cases, the primary activity is considered, in this case programming, rather than the minor one.

Acknowledgments

We would like to thank the participants of our case study, which without them it would not have been possible.

References

[1] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[2] Jennifer A Chatman and Francis J Flynn. The influence of demographic heterogeneity on the emergence and consequences of cooperative norms in work teams. *Academy of Management Journal*, 44(5):956–974, 2001.

[3] Cristina B Gibson and Mary E Zellmer-Bruhn. Metaphors and meaning: An intercultural analysis of the concept of teamwork. *Administrative Science Quarterly*, 46(2):274–303, 2001.

[4] Richard A Guzzo and Gregory P Shea. Group performance and intergroup relations in organizations. *Handbook of industrial and organizational psychology*, 3:269–313, 1992.

[5] David A Harrison, Kenneth H Price, and Myrtle P Bell. Beyond relational demography: Time and the effects of surface-and deep-level diversity on work group cohesion. *Academy of management journal*, 41(1):96–107, 1998.

[6] Jeffrey Pfeffer. Organizational demography. *Research in organizational behavior*, 1983.

[7] W Gary Wagner, Jeffrey Pfeffer, and Charles A O'Reilly III. Organizational demography and turnover in top-management group. *Administrative Science Quarterly*, pages 74–92, 1984.

[8] Don Wells. Extreme programming, 2014. [Online; accessed 06-January-2014].

[9] Katherine Y Williams and Charles A O'Reilly. Demography and diversity in organizations: A review of 40 years of research. *Research in organizational behavior*, 20:77–140, 1998.